

club 1802

P.O. BOX 985
DICKINSON, TEXAS 77539

ISSUE #2 APRIL 1979 PAGE 1

of 43

April 1979

Dear Computerpeople::

Netronics is advertising FULL BASIC for the Elf II. It is to be available this summer. As soon as I get a copy you will be hearing about it. My 500 or so programs in BASIC will then be modified to fit the ELF II.

There has been a delay receiving the binders, mostly my fault trying to make certain there were no typing errors. The front cover contains the instruction set as laid out in the last newsletter. The binders will be shipped out in May.

The surveys started in this issue make no mention of programs available in the magazines. They will be handled separately as BASIC becomes available.

The ELF II with peripherals will be demonstrated at the Johnson Spacecraft Center (JSC) Microcomputer Club on Thurs., June 7, 1979. Those of you in the Houston area might want to attend.

The microCOMputer Club at College of the Mainland is sponsoring a Microcomputer Fair, the second or third weekend in September. I'll keep you posted.

I'll be demonstrating and talking about audiographics at the American Association of Physics Teachers Summer Meeting at Las Cruces, NM. Handwriting by way of a Bit Pad and speech by way of a microphone are simultaneously recorded on separate tracks of an audio cassette. When played back the writing appears on the screen synchronized with the voice. Much better than a televised lecture! A 6800 takes care of everything. Z80 is next and then maybe the 1802.

If you want the game of LIFE contained in this newsletter on paper tape or cassette please write and I'll work out a reasonable price.

Starting next month and for about 3 months, I'll have a Floating Point Arithmetic Package for the 1802 that takes up about 3K of memory. If you can't wait, there are 30-40 pages, xeroxing can get it to you faster, but it would be expensive (roughly 25¢/page). Let me know.

club 1802

P.O. BOX 985

DICKINSON, TEXAS 77539

ISSUE #2 APRIL 1979 PAGE 2

Next month's newsletter will also contain an announcement of a 32K dynamic memory board for the ELF and a version for the S-100 Super ELF.

The kit that solves video problems is available from Henry Claussen, 305 Blackstone, Alvin Tx 77511, for \$10.95.

Till next issue, keep on computing!

John L. Hubisz

JOHN L. HUBISZ

*12 issues plus a
3-ring binder → \$15.
1 ST 8 issues → \$6.
Send check to "club 1802" or use*

extra fast service!

Ship to	
NAME	DATE
STREET ADDRESS	
CITY	STATE ZIP

MASTER CHARGE OR BANK AMERICARD WELCOME	
	COPY FULL ACCOUNT NO. FROM YOUR BANKAMERICARD
	MY CARD EXPIRES:
	COPY FULL ACCOUNT NO. FROM YOUR MASTER CHARGE
	COPY NO. ABOVE YOUR NAME
	MY CARD EXPIRES:
Signature of Card Holder _____	

METHOD OF PAYMENT
☐ CHECK OR MONEY ORDER (ENCLOSED)
☐ BANK CARD (SEE BOX AT LEFT)

LETTERS

I hope to be caught up with all letters by the end of May. I haven't written as many letters in all my life as I have in the past two months.

From now on, most letters will be answered in this column, but when appropriate a personal response will be sent.

Everyone who had been listed by Charles Manry as a subscriber will receive at least some of my newsletters (probably the first seven). The line above your name on the mailing label begins with six numbers. The first four are your membership number and the next two are the issue number with which your subscription expires. If these two digits are "19" then your subscription ends with my issue #19. If for any reason I don't reach the 19th issue your balance will be refunded.

The Disk Controller may be ready within two months. I'll keep you posted.

Micro Tronics is helping to cut my costs. Their advertisements will always be separate from the newsletter and mention of their products in the newsletter will be handled in the same way as everything else anyone in the club brings to my attention. The only software that will be sold will be that which is too long to publish or that which would only be of interest to a few readers. When sold, the author will receive something, the tape will cost something, the postage will cost something and time and handling will cost something. Costs will include these items and no more. I will keep costs as reasonable as possible.

Chuck Wilhite 4613 Kings Drive, Wilmington, NC 28405 writes that he has a VIP with 7K of RAM, 2 HEX keyboards, 4 bit A/D converter with 2 joysticks, CT-64 terminal, LA36 DEC writer, reel-to-reel recorder, parallel ASCII keyboard, and his stereo on remote control (touch tone) via an output port. The CRT monitor doubles for 1861 graphics viewing via an analog switch (4066) controlled by the VIP tv-on and tv-off commands. He runs the chip-8 language, Netronics TINY BASIC and all the games originally designed for RCA's Studio II. (Chuck - send some articles!) He's also working on a serial link from his VIP to a PDP-11/04. (I'm interested in that as I'll have an ELF II up soon and I already have the Terak which is essentially a PDP-11.) Can anyone help Chuck, and me, with a cross-assembler for a PDP-11 to generate 1802 machine language?

P.O. BOX 985

DICKINSON, TEXAS 77539

PASCAL requires a 56K byte machine. You won't see it on an 1802 for a while yet.

A. Rothstein 8500 Irwin Road, Bloomington, MN 55437 writes that he has 1802 cpu chips for \$11.20!

To Don Stevens: the CMOS 4xxx chips are from Siliconix not RCA, but specifications are the same. The User Manual for 1802 by RCA, the Short Course in Programming by Tom Pittman, and a Short Course in TINY BASIC are all \$5.00 from Micro Tronics.

To Mark Marlowe: ALL 8080 programs will run with the 8080 to 1802 conversion program, but it's a few months away yet.

Wilburn M. Lowe Jr. 617 Elm St., Minden, LA 71055 sends us the following:

COSMAC ELF MEMORY TEST PROGRAM

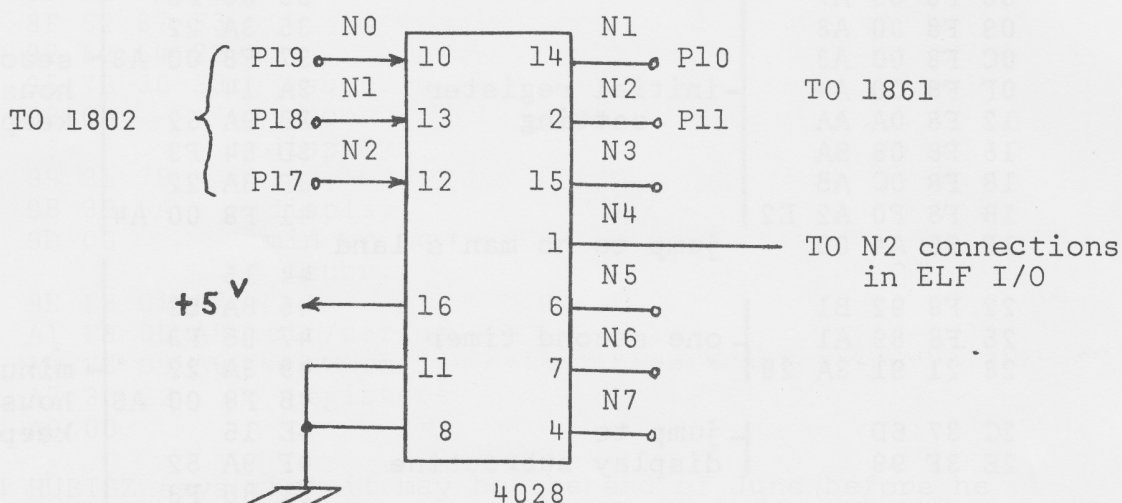
This program tests RAM memory from address 001E up, by storing "55" at address 001E, then retrieving the data byte stored and comparing it with "55". If they are equal, the process is repeated with the next memory location. If the data stored and the data retrieved are not equal, the program stops with the high byte memory address displayed, and the "Q" light comes on. Pressing the "Input" switch will display the low byte memory address, thus locating the address at fault. In order to test all eight bits, the program can be run again, substituting "AA" for "55" at addresses 0009 and 0010.

Address	Instruction	Remarks
0000	F8 00	Load "00" into D
0002	B3	Put D high in register 3.1
0003	F8 1E	Load "1E" into D
0005	A3	Put D low in register 3.0
0006	E3	Set X = 3
0007	13	Increment register.3; R3 + 1
0008	F8 55	Load "55" into D
000A	53	Store D in memory of register 3
000B	93	Get high from register 3.1 to D
000C	23	Decrement register 3; R3 - 1
000D	53	Store D in memory of register 3

Address	Instruction	Remarks
000E	64	Display memory of register 3; R3 + 1
000F	F8 55	Load "55" into D
0011	F3	XOR memory of register 3 with D
0012	32 07	If D = 0 go to 0007
0014	7B	Set Q = 1
0015	83	Get low from register 3.0 to D
0016	23	Decrement register 3; R3 - 1
0017	53	Store D in memory of register 3
0018	3F 18	Wait for Input switch pressed
001A	64	Display memory of register 3
001B	37 1B	Wait for Input switch release
001D	00	Stop

Mike Cohen 2255 Barker Ave, New York, NY 10014 says:

If you run out of I/O parts, this N-line decoder (used in the Super Elf) is a good alternative to the one in this month's "Popular Electronics" (March 1978).



N3 can be used in the ASCII Keyboard interface; change the 69 instruction to 6B.

P.O. BOX 985

DICKINSON, TEXAS 77539

G. Wilson 5319 Ridgecrest Drive, Salt Lake City, UT 84118

DIGITAL CLOCK SIMULATOR PROGRAM

The enclosed program is one I just finished for a Digital Clock Simulator. It is designed for the ELF II and will just fit in the 256 bytes. It gives the Elf something to keep it busy when it is not being used for important things like games.

Simply load the program, load the Hours (1-12) at location 01. Load the minutes (0-59) at location 04.

The program will display Hours for one second then the Minutes for one second. It will turn on Led Q1 to indicate when hours are being displayed. To display seconds simply push the input key and seconds will be displayed as long as it is depressed.

At location 22-2B is the one second timer. I found my Elf was not at calculated frequency, so you may have to play around with the divide bytes at location 23 and 26. If your Elf does not have a 3.58 MHz crystal these bytes will have to be changed. After 3 or 4 tries my clock was within 3 sec. in a 24 hour period.

00 F8 00 A5	load hours	30 13	
03 F8 00 A6	load minutes	31 8A 52	
06 F8 00 A7		33 83 F3	
09 F8 00 A8		35 3A 22	
0C F8 00 A3		37 F8 00 A3	seconds
0F F8 00 A4	initial register	3A 14	house-
12 F8 0A AA	setting	3B 9A 52	keeping
15 F8 06 BA		3D 84 F3	
18 F8 0C AB		3F 3A 22	
1B F8 F0 A2 E2		41 F8 00 A4	
1F 30 AA 00	jump to no man's land	44 15	
22 F8 92 B1		45 8A 52	
25 F8 89 A1	one second timer	47 85 F3	
28 21 91 3A 28		49 3A 22	minute
		4B F8 00 A5	house-
2C 37 6D	jump to	4E 16	keeping
2E 3F 99	display subroutine	4F 9A 52	
		51 86 F3	
		53 3A 22	
		55 F8 00 A6	

club 1802

P.O. BOX 985

DICKINSON, TEXAS 77539

ISSUE #2 APRIL 1979 PAGE 7

58 17					AA 86 AC	
59 19					AC 85 AD	
5A 8B 52					AE 8C	
5C 89 F3					AF F6 F6 F6 F6	
5E 32 9E					B3 A6	
60 8A 52	hours				B4 FE FE FE FE	
62 87 F3	house-				B8 52	
64 3A 22	keeping				B9 8C	
66 F8 00 A7					BA F3 A5	
69 18					BC 8D	
6A 30 22					BD F6 F6 F6 F6	subr.
6C 00					C1 A8	looks at
					C2 FE FE FE FE	time
6D 84					C6 52	entered
6E FE FE FE FE					C7 8D	and sets
72 52 83 F3					C8 F3 A7	all
75 52 64 22					CA 87 A9	registers
78 30 30	subr.				CC 88	to
7A 00	sec				CD 32 D4	correct
	display				CF 8A 52	count
7B 86					D1 89 F4	
7C FE FE FE FE					D3 A9 29	
80 52 85 F3					D5 30 22	
83 52 64 22					D7 00	
86 7A 30 30	subr.					
89 00	min					
	display					
8A 88						
8B FE FE FE FE						
8F 52 87 F3						
92 52 64 22						
95 7B 30 30	subr.					
98 00	hour					
	display					
99 31 7B						
9B 39 8A	display					
9D 00	min or hr					
	subr.					
9E F8 01 A7						
A1 F8 00 A8	clr/set up					
A4 F8 00 A9	hours					
A7 30 22	registers					
A9 00						

JOHN HUBISZ says that it may be the end of June before he catches up with mail. Some letters go back to January 1978!

SHORT NOTES

BOOKS and MAGAZINES are available in libraries. If your library doesn't have a particular book or magazine, ask the librarian to get it. Often they are quite happy to do so because they want their books and magazines to be read. If you can show that at least a few people will read a certain magazine, the librarian will gladly subscribe. Make sure it's read!

CECIL M. BOUIE has a Morse code decoder program for the P O BOX 802 ELF II and equivalent computers. The YORK, PA 17405 signals to be decoded can be taken from the speaker leads or headphone jack. The hardware required to run the program includes an 1861 based video display, 1K of programmable RAM, and a very simple interface circuit. An object listing and a description of the interface can be had by sending \$2.00 and a SASE (#10 envelope or larger, that is 10.5 cm x 24.0 cm or larger).

I receive Kilobaud, Personal Computing, Byte, Creative Computing, Educom, Games, Pipeline, Microcomputing Industry Newsletter, Judge, Bridge, PPC, and a host of other newsletters and magazines. Rest assured that any item of direct interest to 1802 users in these sources will be cited in this newsletter. In fact you may expect mention of those items of indirect interest if they can be adapted to the 1802.

club 1802

P.O. BOX 985

DICKINSON, TEXAS 77539

ISSUE #2 APRIL 1979 PAGE 9

Allen T. Hjelmfelt Jr.

22 E. Craig

Columbia, Missouri 65201

INTRODUCTION

The RCA Cosmac computer provides an excellent low cost entry into computer hardware and programming. The many excellent kits provide the basis for hardware development. Programming, however, is another matter. The owner who lacks experience in machine language programming may well be quite mystified by the commands needed. Programming an operation may be straight forward, but finding and displaying the result is another problem.

This introduction to programming is only intended to scratch the limits of the capabilities of the CDP 1802. It is not intended to make the reader an accomplished programmer. It is intended to lead the reader through the most frequently needed instructions. The instruction set should be comprehensible. Programming is not easy and only practice brings facility, but the frustrations of a new Cosmac owner should be alleviated.

The number system and the structure of the CDP 1802 chip are discussed prior to investigating the instruction set. Instructions are introduced as needed to generate various programs. As a result the instructions are not introduced systematically but as a user needs them. Some details of binary arithmetic and logical operations are given and used to indicate applications of various instructions.

THE NUMBER SYSTEM

Before programming the Elf computer, one must understand the number system with which the machine operates. The computer uses binary numbers, but for input and output hexadecimal is more convenient; so hexadecimal is used. Decimal numbers are the normal form, but are not used. The reasons for use of these number systems will be explored.

Decimal numbers are used in most everyday work. The number 756 may be taken apart by indicating that

7 is in the hundreds place

5 is in the tens place

6 is in the ones place

Another way of expressing the same concept is to write

$$756 = (7 \times 100) + (5 \times 10) + (6 \times 1),$$

or in yet another form

$$756 = (7 \times 10^2) + (5 \times 10^1) + (6 \times 10^0).$$

It is in this last form that one sees the reason for speaking of this system as decimal or base 10.

Note that we call this base 10, but we recognize that the numbers are 0,1,2,3,4,5,6,7,8, and 9. The base number 10 is a combination of these.

The computer works on a binary or base 2 system. This is the easiest form to display, but the most time consuming to use.

The only digits used in this system are 0 and 1. A light off can represent 0 and a light on can represent 1; a switch up can represent 1 and down can represent 0.

The binary number 10011101 means

$$\begin{aligned} 10011101 &= (1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + \\ &\quad (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + \\ &\quad (0 \times 2^1) + (1 \times 2^0). \end{aligned}$$

This expression is binary on the left side, but decimal on the right. The binary number can also be expressed as

$$\begin{aligned} 10011101 &= (1 \times 128) + (0 \times 64) + (0 \times 32) + \\ &\quad (1 \times 16) + (1 \times 8) + (1 \times 4) + \\ &\quad (0 \times 2) + (1 \times 1) \quad \text{or} \end{aligned}$$

$$10011101 = 157$$

Again the left hand side is binary and the right hand side is decimal.

As a matter of interest, the decimal number 9 is expressed as $9 = 1001$ in the binary system.

The computer works on the binary system and works with numbers containing 8 bits or digits. In order to enter a number one must set eight switches. To read the output the pattern of eight lights must be deciphered. This is inconvenient and often leads to errors. It is more convenient to divide the eight digits used by the computer into two groups of four. Thus the previous number 10011101 would be divided into the two groups

P.O. BOX 985

DICKINSON, TEXAS 77539

1001 and 1101, a number system that represents the combinations of four binary digits.

Four binary digits can represent binary numbers from 0000 to 1111. The number 0000 is obviously the decimal number 0. The binary number

$$1111 = (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

$$= 8 + 4 + 2 + 1 \text{ or}$$

$$1111 = 15$$

Thus a group of 4 binary bits represents the decimal numbers from 0 to 15, or form a base 16 number system. It is easy for the computer to convert back and forth from binary to base 16 or hexadecimal. Decimal is still more convenient for use than hexadecimal, but hexadecimal is more convenient than binary.

In order to use hexadecimal, digits are needed to represent the decimal numbers from 0 to 15. The system used is the group:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

The hexadecimal number 9D can be expressed as

$$9D = (9 \times 16^1) + (D \times 16^0)$$

$$= (9 \times 16) + (13 \times 1)$$

$$= 157$$

The hexadecimal number 9D is the same as the binary number 10011101. The convenience of hexadecimal numbers instead of binary should be obvious.

Conversion between decimal, binary and hexadecimal numbers is shown in Table 1.1. The conversion between decimal numbers from 0 to 256, and hexadecimal numbers from 0 to FF is given in Table 1.2.

The computer works with binary numbers. It is more convenient for humans to work with larger base systems. Base 16 is a compromise between machine and human requirements.

TABLE 1.1

EQUIVALENT NUMBERS IN VARIOUS BASES

<u>DECIMAL</u>	<u>BINARY</u>	<u>HEXADECIMAL</u>
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

P.O. BOX 985

DICKINSON, TEXAS 77539

TABLE 1.2 HEX TO DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

THE CENTRAL PROCESSING UNIT AND MEMORY

The central processing unit of the Elf computer is the large scale integrated circuit labled CDP 1802. This is the control element of the computer. Programming consists of writing instructions to be carried out by this element of the computer. The architecture of this element must be known in order to understand the instructions that can be used in programming. The description that follows is limited to that essential for the beginning programmer.

The architecture of the CPU is shown in Fig. 1.1. The primary element of concern is the box marked D. During program operation much of the data passes through this box. The next important set element is a group of 16 registers. These registers can be used to store data or to store addresses of memory locations. The 16 registers hold 16 binary bits, but are divided into upper and lower halves, that can be used independently.

The registers are numbered from 0 to 15, or in hexadecimal from 0 to F. In order to distinguish between the upper and lower halves of the register the following notation is used

Upper half of register C - R(C).1

Lower half of register C - R(C).0

Memory addresses may be maintained in the registers. The programmed instructions are stored in the memory. Normally ,

P.O. BOX 985

DICKINSON, TEXAS 77539

register 0 is used to keep track of the instruction sequence.

The other significant element of the CPU is the box marked X. This box is used to point to one of the registers and therefore contains a number between 0 and F.

This description of the CPU architecture may seem a bit confusing, but should become clearer as one progresses with programming. An example of the use of some of the elements can be shown by use of the instruction "to display a number."

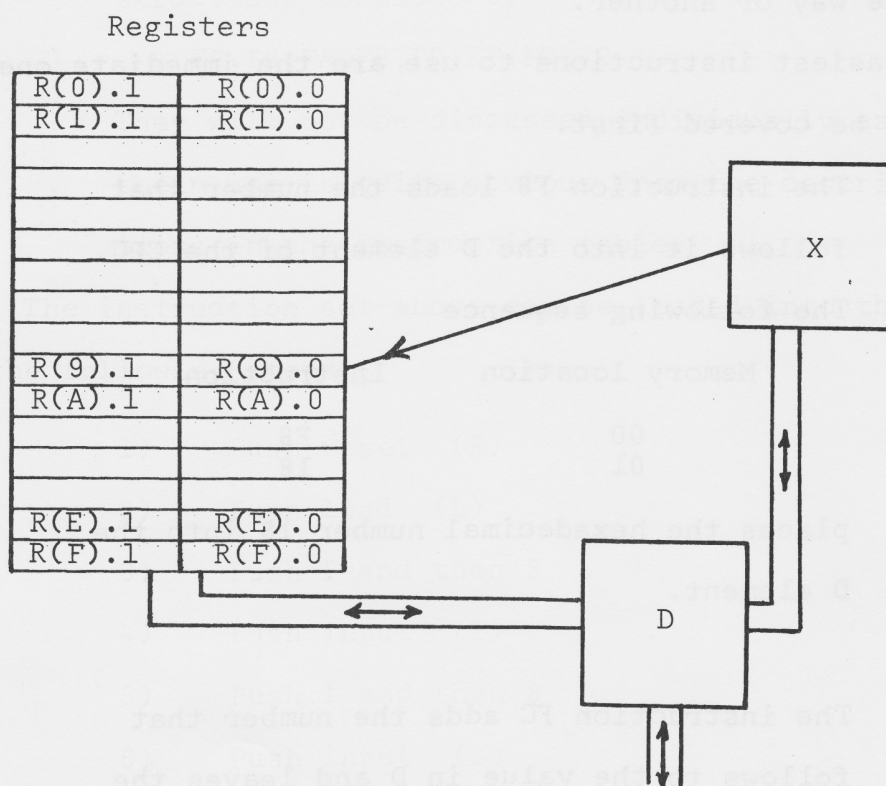
The display instruction is 64. When the CPU reads this instruction it first looks at the X block. Recall that the X block points to a register; so the CPU reads the number in X and then goes to that register. The register contains a memory address. The CPU now goes to that memory address, reads what is stored there and displays the value. This seems like an involved process, but it does work. With involved programs it turns out to be very efficient.

Each of the elements, D, upper and lower halves of the registers, and each location in the memory, stores one word of information. A word consists of eight binary bits or two hexadecimal digits. The element labeled X is an exception to this. It stores only four binary bits or one hexadecimal digit.

The memory of the basic Elf computer has 256 memory locations. These are numbered in hexadecimal notation from 00 to FF. One instruction or piece of data may be stored in each

location. The computer follows the instructions in sequence unless one of the instructions tells the CPU to go to someplace other than the instruction that follows. This will be covered in the section on programming.

Fig. 1.1: A conceptual view of the CDP 1802 architecture.



P.O. BOX 985

DICKINSON, TEXAS 77539

PROGRAMMING THE ELF COMPUTER

The instruction list for the CDP 1802 CPU contains more than 50 commands. Good programming comes from efficient use of these commands. This comes from practice and from an understanding of the details of their operation. At this point, however, good programming is less important than getting some result, one way or another.

The easiest instructions to use are the immediate ones. These will be covered first.

F8 The instruction F8 loads the number that follows it into the D element of the CPU. The following sequence

Memory location	Instruction
00	F8
01	18

places the hexadecimal number 18 into the D element.

FC The instruction FC adds the number that follows to the value in D and leaves the result in D. The following sequence of instructions

Memory location	Instruction
00	F8
01	18
02	FC
03	5F

results in first loading 18 into D. Then 5F is added to the 18 already in D. The result is 77 in hexadecimal and this result stays in D.

Other immediate instructions take care of subtractions, and logical AND, OR, and EXCLUSIVE OR operations. This group of instructions consists of

F8,F9,FA,FB,FD,FF,FC,7C

They will not be discussed individually as their use is quite obvious from the descriptions in the instruction table.

The instruction set above can be loaded into the computer by the following steps

- 1) Push reset (R)
- 2) Push load (L)
- 3) Push F and then 8
- 4) Push input (I)
- 5) Push 1 and then 8
- 6) Push input (I)
- 7) Push F and then C
- 8) Push input (I)
- 9) Push 5 and then F
- 10) Push input (I)

The program can be run by

- 1) Push reset (R)
- 2) Push go (G)

Unfortunately, the result is left in D; so we can't tell yet what happened.

One possibility is to store the result in a memory location, then step through memory to get the result.

5n The instruction 5n stores the value in D into the memory whose address is stored in the register, whose number is represented by n. In practice n is replaced by some number between 0 and F.

The sequence of instructions could now be

00	F8
01	18
02	FC
03	5F
04	56

Through step 03 the process is the same as before. At step 04 the CPU looks at register 6, reads the address and stores the 77 that is in D in that memory location. The problem now is that we do not know what is in register 6, so we do not know where the result is stored.

The instruction An is useful at this point.

An The instruction An stores the number in D in the register represented by n. In practice, n is replaced by a number between 0 and F.

If you wish to store the result of the previous program in location 0B we must get 0A in register 6. Instruction A6 will be used, but first the value 0B must be placed in D. An F8 instruction will be used for this step. The following sequence of instructions will give the desired result.

Memory location	Instruction	Comments
00	F8	load 0B
01	0B	into D
02	A6	put 0A into register 6
03	F8	add
04	18	5F
05	FC	to
06	5F	18
07	56	store value in D into 0B

This can be programmed and run following the procedure

- 1) Reset
- 2) Load
- 3) F8 Input
- 4) 0B Input ...
- 10) 56 Input then
- 1) Reset
- 2) Go

To display the result, push

- 1) Reset
- 2) Memory Protect
- 3) Input

P.O. BOX 985

DICKINSON, TEXAS 77539

and continue pushing Input to read the memory locations in sequence. The first 8 locations displayed will be the program. The locations 08,09, and 0A are meaningless, but the next, 0B should have the value 77.

The program would be even better if the result could be displayed. The display instruction is 64.

64 Element X is checked for a register number. That register is looked to for a memory address. The value in the memory address is displayed.

The memory to be displayed has address 0B and this is stored in register 6. Element X must point to register 6; so a 6 must be loaded into X. The proper instruction is En.

En The number represented by n is loaded into X.
In practice, n is replaced by a number between 0 and F.

The program that was used previously can now be modified to display the result by adding two instructions as follows.

00	F8	
01	0B	
02	A6	
03	F8	
04	18	
05	FC	
06	5F	
07	56	
08	E6	place 6 in X
09	64	display result

This results in a 10 step program that will add two numbers together and display the result. When using this remember

P.O. BOX 985

DICKINSON, TEXAS 77539

that the largest number that can be displayed is FF or 255 in decimal.

At the beginning of this development the use of the immediate instructions was to be shown. In order to see if the operations worked several other instructions were introduced. The instructions used thus far are An, En, F8, FC, 5n, and 64. Compare the description of these instructions in the instruction table with the manner in which they were used in the program.

The other immediate instructions are F9, FA, FB, FD, FF, and 7C. These instructions cover subtraction, and the logical operations of AND, OR and EXCLUSIVE OR. Their use will not be described in detail.

Another useful instruction type is the branch. The instruction 30 causes the processor to go to the address given by the next pair of digits and execute the instruction in that location. Thus,

30
1F

sends the processor to location 1F to locate the next instruction.

The branch instruction can be used to place the number of the memory in the memory location. The following program accomplishes this by numbering backwards.

00	F8	FF in
01	FF	register D
02	A2	FF in register 2
03	82	value in register 2 → D
04	52	store value in D in memory
		no. in register 2
05	22	reduce register 2 by 1
06	30	
07	03	return to instruction 03

After pushing Reset, Go, to run the program, push Reset, Load, Protect and with each push of Input the quantity in memory will appear. The first six steps of the program will appear, and then the 03 in step 7 will be replaced by 07. When the program has worked down to this place and the 03 is replaced by 07 the "30" instruction then directs the program to stop.

NOTE: Allen has a Super Elf from Quest Electronics.

club 1802

P.O. BOX 985

DICKINSON, TEXAS 77539

WHAT'S HAPPENING?

In this column I would like to review a number of journals, magazines, advertisements and the like for articles and

materials of interest to 1802 club members. For the next few issues it would probably be better if the column were entitled "WHAT HAPPENED?"; but, being hopeful that I will catch up to current issues soon, I'll leave it as is.

"KILOBAUD - The Computer Hobbyist Magazine" began publication in January 1977. Issue #29 (May 1979) has over 175 pages (including a letter from Barry Erick, member #1076, and 2 short articles on the COSMAC VIP). The magazine is now called "Kilobaud MICROCOMPUTING". It is generally excellent and contains something for everyone. Prior to January 1977, articles on computers appeared in "73" a HAM magazine. Wayne Green decided that there was enough material for a dedicated magazine for the hobbyist and "KILOBAUD" was born. When reading "73" note all the I/O articles - they're for computer hobbyists.

ISSUE #1 JAN 77

"Tiny Basic ... a mini-language for your micro" by Tom Pittman, pp. 34-37. Tiny Basic is available on the 1802. Tom Pittman's "Experimenter's Kit" demonstrates that Tiny Basic's limitations need not be all that limiting.

"Computer Control of the World!" by Chris Bowick, pp. 62-63. A beginning article on turning AC powered devices on and off with your computer, using an optoisolator.

club 1802

ISSUE #2 APRIL 1979 PAGE 26

P.O. BOX 985

DICKINSON, TEXAS 77539

Two regular features to look for in the early issues are the GLOSSARY and The BASIC Forum. Then starting in Issue #5 there is the excellent series "Kilobaud Klassroom" which is now (Issue #29) up to lesson #18.

ISSUE #2 FEB 77

Under "News of the Industry" p.16, a kit for building a micro-computer based on the CDP 1802 COSMAC microprocessor is available from RCA! A 1 1/2 column announcement. Above the announcement is a shorter one for a Real Time Clock, a useful peripheral for any computer.

"ZAP!" by Joe Magee, pp. 18-20. Static electricity can damage IC's. Here's an article that tells why and also describes what can be done to avoid problems.

"RCA Tries Again ... with the 1802" by William J. Haberhern Jr. pp. 90-94. If more "advertisements" were like this one we would learn a lot more about products and probably feel better about companies. Actually still a useful reference on the 1802.

ISSUE #3 MAR 77

In March, survey results were reported and 14 respondents admitted to having RCA 1802's probably indicating about 150 RCA 1802's out there. A letter (p. 10) from W. J. Haberhern (see issue #2) noting that the UC 1800 was finding more applications in industry than in the hobby market.

ISSUE #4 APR 77

ISSUE #5 MAY 77

"Bridging the Gap ... tips on turning an application into a program" by David E. Stanfield pp. 90-95.

Before writing a program a problem must be solved. Without a clear solution there is no sense in writing the program. All programs should have documentation, or else it will be worthless when you come back to it and find that you have forgotten why you did such-and-such. This article gives some good advice and spends some time on flow-charting the tic-tac-toe problem.

ISSUE #6 JUN 77

ISSUE #7 JUL 77

ISSUE #8 AUG 77

"Trigger Your Oscilloscope ... add value to inexpensive 'scopes" by W. J. Prudhomme pp.34-38.

It so happens that I purchased a second hand Heathkit IO-102 scope a short while ago. This article shows how to construct a triggered sweep circuit for untriggered scopes such as the IO-102. Oscilloscope theory is also well done.

"Random Integer Program ... for games, sorting, or statistics" by Phillip Tubb pp. 46-47.

P.O. BOX 985

DICKINSON, TEXAS 77539

There are many game programs that require such a sub program.

The discussion is helpful but none of the programs can be implemented in Tiny Basic but I'm going to work on it.

"Is the KIM-1 For Every-1?" by Robert M. Tripp, Ph.D. pp. 56-58. Based on the 6502 the KIM-1 is probably the biggest competitor of the 1802 - based microcomputers. More software is available for the 6502. Let's get to work!

"Enter the Audible Computer!" by John E. Stith page 80.

Many uses come to mind for noise from a computer. There is currently a chip available from Texas Instruments that makes all kinds of noises (SN76477).

"Sooo, You Want to be an Author" by George Young pp.90-91.

Hints on how to write articles for magazines ... or newsletters! I'm not as difficult to write for, but the hints are worthwhile.

ISSUE #9 SEP 77

"Build Your Own ASCII Keyboard ... with serial and parallel output" by Robert Brehm pp. 22-27.

I mention this article, not to point out how to build your own, but for the theory included. The better you understand how keyboards operate the easier it is to modify or troubleshoot.

ISSUE #10 OCT 77

Ten more next month and then ten in June and "Kilobaud" will be up to date. "Games" has no mention of the 1802 but there are many suggestions for computer application. I'll start "BYTE" in July.

P.O. BOX 985

DICKINSON, TEXAS 77539

Karl Gerhard 264 Taylor Road, Portsmouth, RI 02871

A SIMPLE INTERRUPT TO AID PROGRAM DEBUGGING

This interrupt feature allows the user to manually cause an interrupt and then display selected general registers. The circuit is trivial; connect a NO pushbutton switch to the interrupt line, pin 36 of the 1802 to ground it when the button is pushed. This generates the interrupt, the rest is software. Then, to display RN.0, enter 8N at the HEXKB; to display RN.1, enter 9N.

I have implemented the SCRT therefore registers 0-7 are used in a standard way. The control program initialization must set R1 pointing to the interrupt routine, and R2 pointing to a temp work byte. With R3 as the "standard" program counter, the interrupt feature easily shows where the CPU is hung or idled. Note, the 1802 RESET does not clear general registers 1-F, therefore you must store addresses in both low and high bytes when setting up the registers. I hope that none of you spent the time I did to realize this! The interrupt routine is as follows:

Wait for input	BN4	\$	00 3F00
	B4	\$	02 3701
	IN4		04 6C
Modify pgm	STR	R1	05 51
Get reg	--		06 (input stored here)
Reg to temp	STR	R2	07 52
Display temp	OUT4		08 64
	DEC	R2	09 22
	B		return0A 3000

Tough program, isn't it?

P.O. BOX 985

DICKINSON, TEXAS 77539

Karl Gerhard 264 Taylor Road, Portsmouth, RI 02871

A SIMPLE INTERRUPT TO AID PROGRAM DEBUGGING

This interrupt feature allows the user to manually cause an interrupt and then display selected general registers. The circuit is trivial; connect a NO pushbutton switch to the interrupt line, pin 36 of the 1802 to ground it when the button is pushed. This generates the interrupt, the rest is software. Then, to display RN.0, enter 8N at the HEXKB; to display RN.1, enter 9N.

I have implemented the SCRT therefore registers 0-7 are used in a standard way. The control program initialization must set R1 pointing to the interrupt routine, and R2 pointing to a temp work byte. With R3 as the "standard" program counter, the interrupt feature easily shows where the CPU is hung or idled. Note, the 1802 RESET does not clear general registers 1-F, therefore you must store addresses in both low and high bytes when setting up the registers. I hope that none of you spent the time I did to realize this! The interrupt routine is as follows:

Wait for input	BN4	\$	00 3F00
	B4	\$	02 3701
	IN4		04 6C
Modify pgm	STR	R1	05 51
Get reg	--		06 (input stored here)
Reg to temp	STR	R2	07 52
Display temp	OUT4		08 64
	DEC	R2	09 22
	B		return0A 3000

Tough program, isn't it?

MACHINE-LANGUAGE GAME OF LIFE PROGRAM FOR COSMAC 1802

by Ben Hutchinson

The game of LIFE was invented by mathematician John Horton Conway in 1970. Though not conceived as a computer game originally, it quickly became popular as such. Surely it is one of the more delightful and absorbent of computer games, especially with a CRT display. Although it can be and is played with a keyboard/printer terminal, it is essentially graphical in nature rather than verbal or numeric, and really needs none of the usual keyboard interchange ("The Klingons are coming! Enter move"). All these things suggest that it might be ideally suited to a COSMAC - based microcomputer equipped with an 1861 video interface chip and nothing more than a hexadecimal keyboard. This turns out to be not only true, but easier than you might think. The accompanying program uses only 1024 bytes (4 pages) of memory, of which half is used for current and alternate display refresh for the standard 32 X 64 display format. The program takes about 2.5 seconds per generation. It was developed and run on an RCA COSMAC VIP, but can easily be adapted to run on any of the ELF or similar small COSMAC systems so long as they have an 1861, 1K or more of memory, and some way to write arbitrary bytes to selected addresses anywhere in memory.

A complete description of the Game of LIFE may be found in Scientific American for October, 1970, page 120. Important follow-up articles appeared as follows: November 1970; January, February, March and April 1971; November 1971; and January, 1972. The name derived from an analogy with the growth of colonies of living organisms, such as bacteria. The idea is to start with a simple pattern of live "cells" (in a COSMAC, 1's in memory giving spots of light) and see how it grows when certain very simple generic growth laws are applied. Every cell in a square array has 8 "neighbors" (counting diagonals). The rules are: Each cell with two or three neighbors survives. Each cell with four or more neighbors dies from overcrowding. Each cell with one neighbor or none dies from isolation. Each empty space with exactly three neighbors has a birth (live cell in next generation). All births and deaths occur simultaneously; together they produce a new pattern for the next "generation". The changes are unusual, sometimes beautiful, always unexpected. Very simple initial patterns can grow to enormous complexity and persist for hundreds of generations.

In the program, two pages of RAM are used for display refresh storage. One stores the current generation's pattern, and is accessed by the interrupt service routine, using R0 as the DMA pointer, as controlled by interrupt and DMA requests from the 1861 video chip. The other page stores the next generation's pattern, as it is built up bit by bit by the program, which of course applies the rules listed above, counting the number of live neighbors for each cell in the current generation. The way the 1861 and 1802 work together allows both display refresh and access to data for testing to go on separately and independently, without of course affecting the display of the current generation. When all cells of the current generation have been tested, and a new generation is ready in the alternate page, the "current" and "new" page addresses are simply swapped between the appropriate R registers, so that the RAM page containing the new pattern begins to refresh the display.

Besides the basic loop which decides and writes the new-generation pattern and periodically swaps display pages, the program contains an initialization routine and a routine to clear the first-used display page. The only other functions needed are the interrupt-servicing routine for display refresh and some way of conveniently entering 1's in selected locations of the first display page, such as from a hex keyboard. In the VIP, these last two tasks are easily done

by routines contained in the ROM operating system. For an ELF or similar machine, all that is needed is to add in RAM the interrupt service code similar to that on the 1861 data sheet or in the PIXIE articles (July-August '77 Popular Electronics), plus some sort of keyboard monitor/operating system such as EHOPS (or even ETOPS) described in Pop. Elec. March 1977, or the ELF-II monitor program, March 1978 Pop. Elec. It will still all fit in 1k, even if the keyboard and interrupt routines are not in ROM.

In order to understand how the heart of the program works (i.e., that part that counts the number of live neighbors which each cell has, and applies the rules of LIFE), refer first to Fig. 1. This is a map of 9 words in the display-refresh page of RAM, with their addresses relative to the center word shown. They are drawn the same way that the 1861 draws their contents on the CRT: in ascending address order, words are painted left to right in 8-word rows, with the first row across the top of the screen. In the description and listing, the word in which bits (cells) are currently being checked is called the "A" word. The "A" word address is stored in RA. The eight neighbor words shown in Fig. 1 are obviously those having one or more neighbor bits of bits in the "A" word, and thus involved in testing that word. These words are referred to by their relative addresses; e.g., "the (A+9) word", not to be confused with the (A+9) address, sometimes called "the address of the (A+9) word".

Figure 2, also drawn the same way the 1861 draws on the CRT with the MSB on the left, shows the designation used for individual bits in each 8-bit word, i.e., the hex value of a word in which only that bit is a "1". This notation is convenient and is used throughout the listing; it corresponds to the hex value of a pointer or mask which picks out the single bit which it designates.

Brief study of figs. 1 and 2 together with the rules of LIFE given above make clear several key facts on which the structure and operation of the COSMAC program are based:

- 1). All neighbors of the center 6 bits of the "A" word (02 thru 40) are contained in only 3 RAM words: A, (A-8), and (A+8).
- 2). In contrast, the 80 and 01 bits (MSB and LSB) clearly require separate and more complex treatment, since in each case three additional RAM words contain at least one neighbor bit, for a total of six words.
- 3). Testing all the neighbors of the 40 bit in the A word is particularly straightforward: all the neighbor bits are in three words, and all can be easily shifted left into DF where a single BNF instruction tests 0 or 1.
- 4). Special provisions must be made when the A word is at the edge of the display area, and one or more words containing neighbor bits do not exist on the display page.

These facts are the key to the organization of the program. As shown in the flow chart of Fig. 3, there are three main code segments concerned with the actual testing of neighbors of all 8 bits in a particular "A" word: one for testing the 80 bit, one for testing the center 6 bits (02 through 40) which uses a loop, and one for testing the 01 bit. Each of these code segments loads the A word and its neighbors from RAM, after testing whether A is an edge word. It then shifts the A, (A-8), and (A+8) words so that the particular bit being tested, with its neighbors, is shifted to occupy the 40 bit position. The shifted words are stored in R6.1, R7.0, and R7.1. For the 80 and 01 bits, single neighbor bits from RAM words to the left and right respectively

of the A word in Fig. 1, are brought in via DF as part of this process. Finally, all three code segments call a single subroutine, SUB 40, which applies the rules of LIFE to the 40 bit in the word in R6.1 and its neighbors in R7.0 and R7.1, as described under (3) above. SUB 40 then writes 0 or 1 in the actual bit position of the A word on the "next-generation" display page. It knows where to write by using the "A alternate" word address in RF, and a pointer to the actual tested bit in RE.0. It knows what to write by testing R6.0, which it used to count the number of 1's among the 8 neighbor bits.

The remaining code consists of rather straightforward and obvious controls to increment and reset the A word address after all bits in a word have been tested, and to swap display pages when the pattern for a new generation has been completely written. There is also a simple subroutine which merely performs 5 left shifts (SUB 5). It is used in testing the "01" bit.

The handling of edge words can be done several ways. One published BASIC version of LIFE simply stops the program whenever the rules would call for a live cell outside the displayed area. In this program, zeros are entered in the working registers (R6 and R7) whenever the A word touches an edge, in place of whatever might be in nonexistent neighbor words. This has the effect of killing or distorting growth when a pattern hits an edge, but at least the run continues and one can watch other parts of the pattern develop. Tests are made for top or bottom edge before testing the 80 bit. Such words are labelled by making RC.1 or RD.0 respectively zero; otherwise, these registers are loaded with the address (A-8) and (A+8) respectively. Left-edge words are checked before test of the 01 bit.

With a VIP, operation is as follows. The program is loaded with a CO (LBR) at location 0027. When the switch is first thrown to RUN, initialization is followed by a jump to the erase routine at 0160, which clears page 6 and calls the operating system. Best next step is to immediately write a C8 (NLBR) in 0027, so that the patterns you are about to enter on page 6 don't get erased when you forget! Then, call the operating system in the normal way (key C held down while switching to RUN) and use the memory write mode to enter the starting pattern somewhere on page 6 (repeat this step as needed). The four center words are 067B, 067C, 0683, and 0684; making a chart helps. Finally switching to RUN with no keys held and a C8 at 0027 starts the evolution, which stops only with a stable pattern or a switch to RESET. With 1 K, use pages 2 and 3; the listing shows how.

The operation can be made less clumsy in a number of ways, like automatic or key-dependent clearing of the erase flag. However, if you want this, program it before you play the game, or you'll be having too much fun to bother, like I did! An ELF, with its hardware-switch mode selection and ability to start execution anywhere in memory, may be able to avoid clever or dependent branching entirely by selecting starting addresses for erase, monitor, or run.

The R registers (all of them) are used as follows:

- RO: Initial program counter, then DMA refresh (initialized by interrupt routine to beginning of display page in use)
- R1: Interrupt service program counter. For VIP, initialize to 8146 before turning on the 1861 chip with a 69 instruction. For ELF or other, initialize to entry point of interrupt routine.
- R2: Stack pointer. Used only by interrupt routine in this program. Initialize

to 01FF, or any place with 2 free bytes.

R3: Main program counter (immediate handover from R0)

R4: SUB-40 subroutine program counter

R5: SUB-5 subroutine program counter

R6.0: 1's counter, used by SUB-40 to count 1's among the eight neighbor bits

R6.1: Working storage for the shifted "A" word being tested

R7.0: Working storage for the shifted (A-8) word being tested

R7.1: Working storage for the shifted (A+8) word being tested

R8: Changed by VIP interrupt routine. Do not use unless using a different

R9: routine which does not increment them

RA: Address of "A" word. INC or DEC for A+1 or A-1 words

RB.1: Display page pointer. VIP interrupt routine loads it into R0.1 before starting DMA display refresh.

RB.0: Used by VIP interrupt routine for scratch pad

RC: Address (A-8) if A is not top-edge word; C.1=00 if top edge

RD: Address (A+8) if A is not bottom-edge; D.0 =0 if bottom edge

RE.0: Single bit=1 to point to bit being tested in "A" word. Center-6-bits loop exited when pointer reaches 01 bit.

RF: Address of "A" word on "other" (next-generation) display page

(A - 9)	(A - 8)	(A - 7)
(A - 1)	"A" WORD	(A + 1)
(A + 7)	(A + 8)	(A + 9)

Fig. 1 Array of memory bytes displayed by 1861, showing RAM addresses relative to center word.

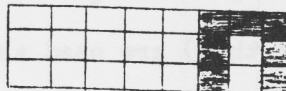
Data Address

03 066D The R Pentomino
06 0675
02 067D

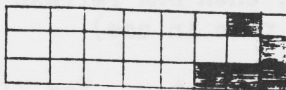
Some LIFE forms, and how to enter them in memory



07 0673 "Pi"
05 067B
05 0683



02 0673 The Glider
01 067B
07 0683



1F 0673 Pulsar CP 48-56-72
11 067B



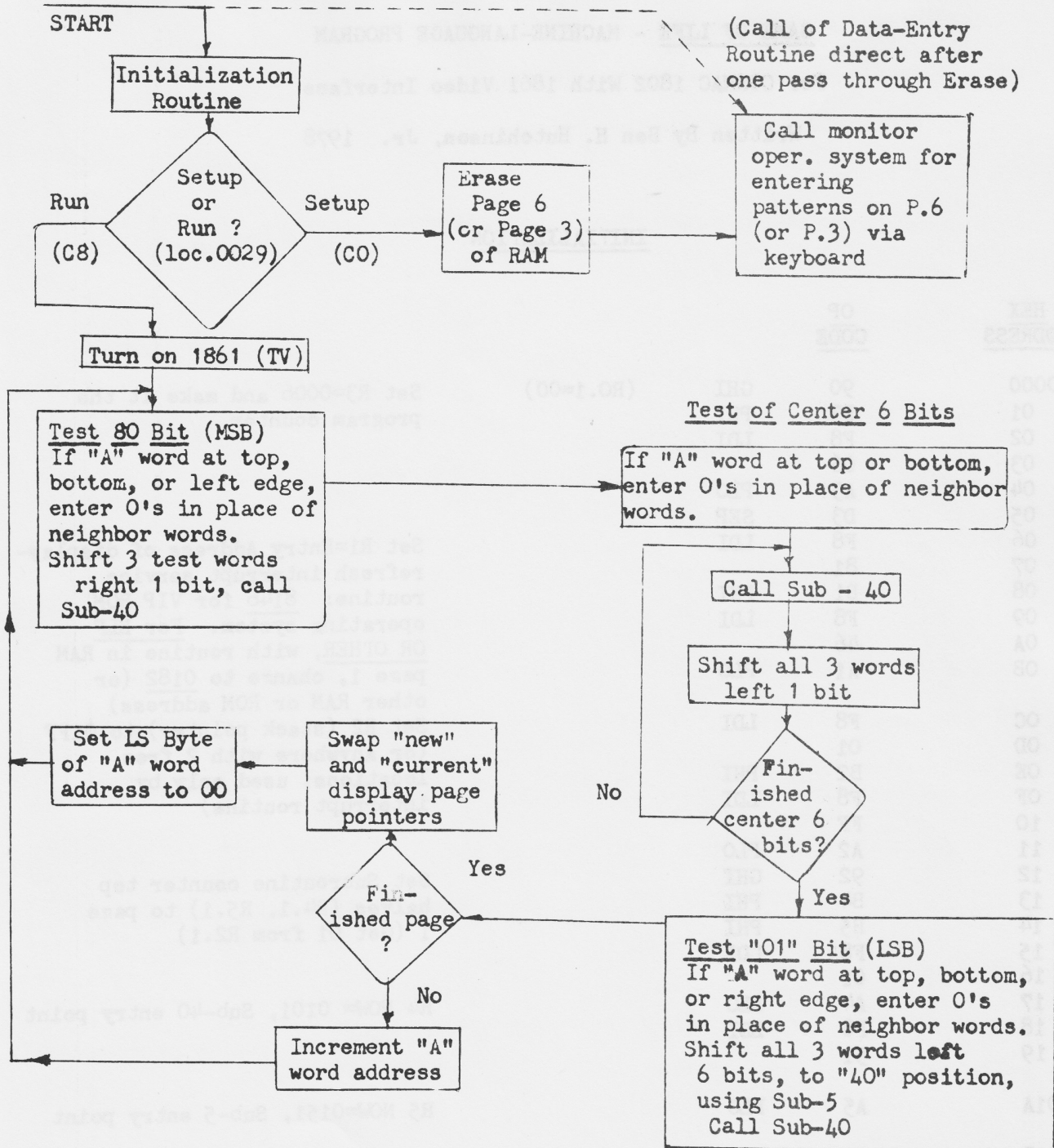
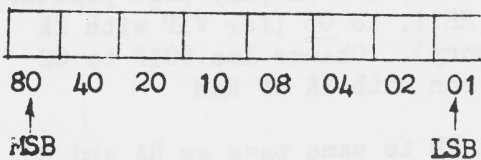


Fig. 3



(A - 8)

A

(A + 8)

80	40	20	
80	40	20	
80	40	20	

Fig. 2 Designation of individual bits in a byte, and the 8 neighbors of the 40 bit.

GAME OF LIFE - MACHINE-LANGUAGE PROGRAM

For COSMAC 1802 With 1861 Video Interface

Written By Ben H. Hutchinson, Jr. 1978

INITIALIZATION

<u>HEX ADDRESS</u>	<u>OP CODE</u>			
0000	90	GHI	(RO.1=00)	Set R3=0006 and make it the program counter
01	B3	PHI		
02	F8	LDI		
03	06			
04	A3	PLO		
05	D3	SEP		
06	F8	LDI		
07	81			Set R1=Entry Address of display-refresh interrupt service routine: 8146 for VIP ROM operating system. For ELF OR OTHER, with routine in RAM page 1, change to 0182 (or other RAM or ROM address)
08	B1	PHI		
09	F8	LDI		
0A	46			
0B	A1	PLO		
0C	F8	LDI		
0D	01			Set R2 (stack pointer) to 01FF (or anywhere with 2 free locations; used only by interrupt routine)
0E	B2	PHI		
0F	F8	LDI		
10	FF			
11	A2	PLO		
12	92	GHI		
13	B4	PHI		Set Subroutine counter top halves (R4.1, R5.1) to page 1 (Get 01 from R2.1)
14	B5	PHI		
15	F8	LDI		
16	01			
17	A4	PLO		
18	F8	LDI		R4 NOW= 0101, Sub-40 entry point
19	51			
001A	A5	PLO		
				R5 NOW=0151, Sub-5 entry point
001B	90	GHI	RO(=00)	Set RA.0 and RF.0=00
1C	AA	PLO		
1D	AF	PLO		
1E	F8	LDI		
001F	06			Set "A" - word address page in RA.1, and display-page pointer in RB.1, to 06 (for VIP with 2K memory). Change loc.001F to 02 to run with 1K of RAM
20	BA	PHI		
21	BB	PHI		
22	C4	NOP	Spare	
23	BD	PHI		
24	F8	LDI		
0025	07			Set RD to same page as RA and RB. Set alternate- page pointer (RF) to 07 (for VIP with 2K memory)
26	BF	PHI		Change loc. 0025 to 03 for 1K of RAM

0027	CO/C8		CO causes branch to erase routine
28	01	(LBR/NLBR)	<u>At 0160; Change to C8 to RUN</u>
29	60		
2A	C4	NOP (3cycles) Spare	
2B	C4		
2C	C4		
2D	E2	SEX	
002E	69	INP 1	<u>Turn on 1861 T.V. chip (Special to</u>
002F	00	IDL	<u>VIP; ELF or other may turn on 1861</u>
			<u>a different way.)</u>

TEST OF "80" BIT (MSB)

0030	F8	LDI	
31	00		
32	BC	PHI	Set RC.1, RD.0, and R7 to default
33	AD	PLO	state (all zero)
34	A7	PLO	
35	B7	PHI	
36	8A	GLO	If low byte of "A" address 08, then
37	FF	SMI	skip, because A is at top edge
38	08		(Leaves RC. 1=0)
39	3B	BNF	
003A	3F		
3B	AC	PLO	Since "A" <u>Not</u> at top edge, Store (A-8)
3C	9A	GHI	in RC. Get display page in use
3D	D3	SEP 3	(2-cycle NOP) SPARE
3E	BC	PHI	Store display page in RC.1
003F	8A	GLO	If low byte of "A" address 08,
40	FF	SMI	then skip, because A is at bottom
41	F8		edge (Leaves RD.0=0)
42	33	BGE	
0043	48		
44	8A	GLO	Since "A" <u>not</u> at bottom edge,
45	FC	ADI	calculate (A+8) and store in RD.
46	08		
47	AD	PLO	
0048	8A	GLO A	If the least-significant 3
49	FA	ANI	bits of the "A"-word address =000,
4A	07		then branch to 006F, because "A"
4B	32	BZ	is at left edge.
004C	6F		
4D	2A	DEC	Since "A" is not at left edge,
4E	4A	LDA	load the (A-1) word, and shift its LSB
4F	76	RSHR	into DF
0050	0A	LDN	Load the "A" word
51	76	RSHR	Shift the neighbor bit in DF to the
			80 position
52	B6	PHI	Store shifted "A" word (test bit
			now at 40 position)

53	9C	GHI	If RC.1=0, then skip shifting (A-8) word, because "A" is at top edge.
54	32	BZ	
0055	5C		
56	2C	DEC	Since "A" <u>not</u> at top edge, load (A-9) word.
57	4C	LDA	Shift LSB into DF
58	76	RSHR	Load (A-8) word
59	0C	LDN	DF → 80 (MSB)
5A	76	RSHR	Store shifted (A-8) word
5B	A7	PLO	If RD.0=0, then skip shifting (A+8) word, because "A" is at bottom edge.
005C	8D	GLO	
5D	32	BZ	
5E	68		
5F	2D	DEC	Since "A" is <u>not</u> at bottom edge, load (A+7) word
60	4D	LDA	Shift LSB into DF
61	76	RSHR	Load (A+8) word
62	0D	LDN	DF → 80 (MSB)
63	76	RSHR	Store shifted (A+8) word
64	B7	PHI	2 - cycle NOPS. SPARES
65	D3	SEP	
66	D3		
67	D3		
0068	F8	LDI	Set Bit Pointer/shift marker to 80
69	80		
6A	AE	PLO	
6B	D4	SEP	Call SUB-40 to count neighbor 1's & write in new page.
6C	D3	SEP	2 - cycle NOP SPARE
6D	30	BR	Go on to test of center 6 bits
6E	80		
006F	0A	LDN	Load "A" word (A is at left edge)
70	F6	SHR	Move 80 Bit → 40 (New 80 bit=0)
71	B6	PHI	Store shifted "A" word
72	9C	GHI	If RC.1=0, then skip, because A is at top edge.
73	32	BZ	
0074	78		
75	0C	LDN	Load (A-8) word from RAM
76	F6	SHR	Move 80 bit → 40 (New 80 bit=0)
77	A7	PLO	Store shifted (A-8) word
0078	8D	GLO	If RD.0=0, then skip, because A is at bottom edge.
79	32	BZ	
007A	7E		
7B	0D	LDN	Load (A+8) word
7C	F6	SHR	Move 80 bit → 40 (New 80 bit=0)
7D	B7	PHI	Store shifted (A+8) word
007E	30	BR	Go to 0068, to set bit pointer and call SUB-40
007F	68		

Test of Center 6 Bits

0080	0A	LDN	Load word to be tested ("A-word")
81	B6	PHI	via RA, and put it in R6.1.
82	9C	GHI	If RC.1 $\neq$ 0, then fetch (A-8) word
83	32	BZ	via RC and put it in R7.0, for "A"
0084	87		is not at top edge; else skip to
85	0C	LDN	0087, leaving R7.0=00.
86	A7	PLO	
0087	8D	GLO	If RD.0 $\neq$ 0, Then fetch (A+8) word
0088	32	BZ	via RD and put it in R7.1, for A
0089	8C		is not at bottom edge; else skip
008A	0D	LDN	to 008C, leaving R7.1=00.
008B	B7	PHI	
008C	F8	LDI	Set the bit-being-tested pointer to 40
8D	40		
8E	AE	PLO	
008F	D4	SEP	Call SUB-40 (Count 1's among 8
			neighbors of bit in "40" position of
			"A" word, write 0 or 1 in new
			generation; i.e., in alt. display
			page).
90	96	GHI	Shift all three words left one place
91	FE	SHL	(A, (A-8), and (A+8) words, in
92	B6	PHI	R6.1, R7.0, & R7.1)
93	87	GLO	
94	FE	SHL	
95	A7	PLO	
96	97	GHI	
97	FE	SHL	
98	B7	PHI	
99	8E	GLO	Move the bit-being-tested pointer one
9A	F6	SHR	place right
9B	AE	PLO	
9C	FB	XRI	Until bit pointer=01, loop to 008F
9D	01		to test next bit from the center 6.
9E	3A	BNZ	Then, move on to "test of 01 bit"
009F	8F		

Test of "01" Bit (LSB)

00A0	D3	SEP	(2-cycle NOP'S) SPARES.
A1	D3		
A2	D3		
A3	D3		
A4	D3		
A5	8A	GLO	If the least - sig. 3 bits of the "A"
A6	FA	ANI	Address in RA.0 are all 1, then skip
A7	07		getting the MSB'S of the (A+1), (A+9),
A8	FB	XRI	and (A-7) words, because "A" is a
A9	07		right edge word.
AA	32	BZ	
00AB	CC		
AC	1A	INC	Increment RA, to address of (A+1) word
AD	0A	LDN	Load (A+1) word
AE	7E	RSHL	MSB $\rightarrow$ DF
AF	2A	DEC	RA="A" word address
BO	0A	LDN	Load "A" word

B1	7E	RSHL	Move MSB of (A+1) word into LSB of D
B2	D5	SEP	Call Sub-5 (5-SHL'S)
B3	B6	PHI	Store shifted "A" word, with bit from (A+1) word
B4	9C	GHI	Skip getting (A-7) & (A-8) words if RC.1=0, showing "A" is top edge
B5	32	B3	
B6	BF		
B7	1C	INC	
B8	0C	LDN	Load (A-7) word
B9	7E	RSHL	MSB → DF
BA	2C	DEC	Load (A-8) word
BB	0C	LDN	
BC	7E	RSHL	DF → LSB
BD	D5	SEP	Call Sub-5
BE	A7	PLO	Store shifted (A-8) word, with bit from (A-7) word
00BF	8D	GLO	If RD.0=0, Skip load of (A+8) and (A+9) words, because "A" -word is at bottom edge.
00C0	32	B3	Load (A+9) word
00C1	DF		
C2	1D	INC	
C3	0D	LDN	
C4	7E	RSHL	Put MSB → DF
C5	2D	DEC	Load (A+8) word
C6	0D	LDN	
C7	7E	RSHL	Put DF → LSB
C8	D5	SEP	Call Sub-5 (5 - SHL)
C9	B7	PHI	Store shifted (A+8) word, with bit from (A+9) word
CA	30	BR	Branch around code for right-edge-word
00CB	DF		Shift of LSB
00CC	0A	LDN	Load "A" word: Begin shift of 01 bit of right-edge-words to make 6 total shifts; move 01 bit to 40.
CD	FE	SHL	Call Sub-5 (5 left shifts)
CE	D5	SEP	
CF	B6	PHI	
DO	9C	GHI	If RC.1=0, then skip load of (A-8) word, because A-word is at <u>top edge</u> . (leaves R7.0=0)
D1	32	B3	Fetch (A-8) word
D2	D7		6 left shifts; 01 bit to 40.
D3	0C	LDN	
D4	FE	SHL	
D5	D5	SEP	
D6	A7	PLO	Store shifted (A-8) word in R7.0
D7	8D	GLO	If RD.1=0, then skip load of (A+8) word, because A-word is at <u>bottom edge</u> (leaves R7.1=0)
D8	32	B3	Load (A+8) word
D9	DF		6 left shifts; 01 bit to 40
DA	0D	LDN	
DB	FE	SHL	
DC	D5	SEP	
DD	B7	PHI	Store shifted (A+8) word
DE	D3	SEP	2-cycle NOP. SPARE
00DF	D4	SEP	Call Sub-40 (Count 1's among 8 neighbors of bit in 40 position, write 0 or 1 in new generation.

Address-Incrementing
and
Page-Alternation Control

00E0	D3	SEP	2-cycle NOP'S. SPARES
E1	D3		
E2	D3		
E3	D3		
E4	8A	GLO	
E5	FB	XRI	If A.0=FF, Last word on current display page has been tested, so go to page alternation; else increment the addresses, and loop.
E6	FF		
E7	32	BZ	
00E8	ED		
E9	1A	INC	RA now has address of new word in current generation
EA	1F	INC	RF now has address of new word in next generation
EB	30	BR	Loop to "Test of 80 Bit", start testing bits in new word
EC	30		
00ED	9F	GHI	Page alternation: Get new page number from RF set A, (A+8) word addresses to new page
EE	BA	PHI	Get "old" page address, new "alternate" page
EF	BD	PHI	Set RF.1 to alternate page (next generation)
FO	9B	GHI	Get new disp. page address
F1	BF	PHI	Set display-page pointer to new address.
F2	9A	GHI	Set A.0, F.0, and C.1 to 00.
F3	BB	PHI	
F4	F8	LDI	
F5	00		
F6	AA	PLO	
F7	AF	PLO	
F8	BC	PHI	
00F9	30	BR	
00FA	30		Loop to "Test of 80 Bit", start testing bits in first word of new page.
FB	D3	SEP	2-cycle NOP'S. SPARES.
FC	D3		
FD	D3		
FE	D3		
00FF	D3		
Subroutine "Sub-40": Tests all eight neighbors of the bit in the "40" position of the shifted "A" word (stored in R6.1), counts the 1's, (count in R6.0), and writes 0 or 1 in the correct bit of the correct word in the "next-generation" page, according to "life" rules. Shifted "neighbor" words (A+8) and A-8) are fetched from R7.1 and R7.0 for the test.			
0100	D3	SEP	Return to main program
0101	F8	LDI	(Entry Point)
02	00		Reset 1's counter (R6.0) to 00
03	A6	PLO	

04	96	GHI	Fetch shifted "A" word
05	FA	ANI	Mask out all but "neighbor" bits,
06	AO		80 and 20.
07	FE	SHL	Shift "80" bit into DF
08	3B	BNF	Increment 1's count if "80" bit
0109	OB		was 1, else skip
010A	16	INC	
010B	FE	SHL	Shift "20" bit into DF
010C	FE	SHL	
010D	3B	BNF	Inc. count if "20" bit was 1, else skip.
010E	10		
010F	16	INC	
0110	97	GHI	Fetch shifted (A+8) word
0111	FE	SHL	"80" bit → DF
0112	3B	BNF	Inc. count if "80" bit was 1, else skip
0113	15		
0114	16	INC	
0115	FE	SHL	Shift, test, count-if-1, "40" bit of
0116	3B	BNF	(A+8)
0117	19		
0118	16	INC	
0119	FE	SHL	20 bit → DF
011A	3B	BNF	Test, count-if-1
011B	1D		
011C	16	INC	
011D	87	GLO	Load shifted (A-8) word
011E	FE	SHL	Shift, test, count-if-1 the three
011F	3B	BNF	"neighbor" bits in the (A-8)
0120	22		word
0121	16	INC	
0122	FE	SHL	
0123	3B	BNF	
0124	26		
0125	16	INC	
0126	FE	SHL	
0127	3B	BNF	
0128	2A		
0129	16	INC	
012A	D4	SEP	2-cycle NOP'S SPARES
2B	D4		
2C	D4		
2D	D4		
2E	D4		
2F	D4		
0130	96	GHI	Get shifted "A" word, mask out all
31	FA	ANI	but "40" bit, i.e., bit whose fate is
32	40		to be decided if bit=0, skip test for
33	32	BZ	2 neighbors
0134	3A		
35	86	GLO	Since "40" bit =1, get 1's count and
36	FB	XRI	branch to "write-1" routine if 1's
37	02		count=2 (2 neighbors; survival)
38	32	BZ	
0139	4A		
013A	86	GLO	Get 1's count

013B	FB	XRI	If # of 1's (neighbors)=3, branch to "write-1" routine (survival or birth); else continue to "write-0" routine: Load bit pointer into D (1 at bit position) complement bit pointer (0 at bit position)
3C	03		
3D	32	BZ	
013E	4A		
3F	8E	GLO	"AND" the complemented bit pointer with word at loc. A in "new generation" page, writing "0", and store in loc.A, "new" page. Branch to Sub-Return point
40	FB	XRI	
41	FF		
42	EF	SEX	
43	F2	AND	2-cycle NOP'S. SPARES
44	5F	STR	
45	30	BR	
0146	00		
47	D4	SEP	"Write-1" routine: Load Bit pointer "OR" with word at loc. A in "new generation" page, writing "1", and store in loc.A, "new" page. Branch to Sub-Return point
48	D4		
49	D4		
014A	8E	GLO	
4B	EF	SEX	
4C	F1	OR	
4D	5F	STR	
014E	30	BR	
014F	00		

"SUB-5" Subroutine: Shifts D Left 5 Times

0150	D3	SEP	Return to Main
0151	FE	SHL	Five left shifts, to move A bit from 02 position to 40 position in word.
52	FE		
53	FE		
54	FE		
55	FE		
56	30	BR	Branch to Sub.Return point
0157	50		

(Note: 0158 to 015F are SPARE locations. Can fill with NOP'S)

Routines to clear display-refresh page and call operating system to allow entry of initial "Life Forms". (VIP operating system call shown)

0160	F8	LDI	Set RA.0=FF (RA.1 has been set earlier to number of initial display page to be cleared)
61	FF		
62	AA	PLO	Store 00 in location M(R(A)), and decrement RA.
63	EA	SEX	
0164	F8	LDI	
65	00		
66	73	STXD	
67	8A	GLO	If RA.0≠FF, page not completely cleared; loop to 0164 and continue clearing
68	FB	XRI	
69	FF		
6A	3A	BNZ	
016B	64		
016C	1A	INC	Increment RA back to (X)00 from (X-1)FF.
016D	F8	LDI	Initialize R-registers as required and call operating system so that Initial "Life Form" pattern data can be entered. -- The code listed after
6E	07		Set R1=07FF
6F	B1	PHI	
70	F8	LDI	

71	FF	
72	A1	PLO
73	F8	LDI
74	80	
75	B2	PHI
76	F8	LDI
77	28	
78	A2	PLO
0179	D2	SEP

Loc. 016D calls the ROM Operating System of a COSMAC VIP with 2K Memory. (For bigger VIP'S, set R1 to top of top RAM page). For other COSMAC systems (ELF, ELF-II, DUPER-ELF, TEKTRON, ETC.), code beginning at 016D should be replaced by code (or by a branch to code) which allows entry of address and data from a keyboard, and writes the data in the specified address on the initial display page.

Set R2= 8028

BR. to VIP ROM oper. system at 8028

Optional Display-Refresh Interrupt-Service Routine
(If Not Using VIP ROM Routine)

0180	42	LDA
0182	C4	NOP
83	22	DEC
84	78	SAV
85	22	DEC
86	52	STR
87	9B	GHI
88	B0	PHI
89	F8	LDI
8A	00	
8B	A0	PLO
8C	C4	NOP
8D	C4	NOP
018E	E2	SEX
8F	80	GLO
90	E2	SEX
91	20	DEC
92	A0	PLO
93	E2	SEX
94	20	DEC
95	A0	PLO
96	E2	SEX
97	20	DEC
98	A0	PLO
99	3C	BN1
019A	8E	
019B	30	BR
019C	80	

Get numbers of display page in use from RB.1

Load display-page number into RO, DMA pointer

NOTE: With the exception of lines 0087 and 0088, this is essentially the 32 X 64 routine on the 1861 data sheet, which see for comments. The change allows this routine to be in ROM and yet display any page.

NOTE: To use this routine, change initialization routine as follows: Loc. 0007=01, Loc.000A=82